A Comparison of Word Embedding Techniques for Similarity Analysis

Tyler Gerth

A Comparison of Word Embedding Techniques for Similarity Analysis

An Undergraduate Honors College Thesis

in the

Department of Computer Science and Computer Engineering

College of Engineering

University of Arkansas

Fayetteville, AR

October, 2020

by

Tyler Gerth

Abstract

There have been a multitude of word embedding techniques developed that allow a computer to process natural language and compare the relationships between different words programmatically. In this paper, similarity analysis, or the testing of words for synonymic relations, is used to compare several of these techniques to see which performs the best. The techniques being compared all utilize the method of creating word vectors, reducing words down into a single vector of numerical values that denote how the word relates to other words that appear around it. In order to get a holistic comparison, multiple analyses were made, with the WOVe technique performing the best overall at producing both the most synonyms and the most accurate synonyms.

1. Introduction

1.1 Background

The process of word embedding, or the reduction of text into real number vectors, has long been one of the primary methods used to perform natural language processing. A word embedding allows researchers to reduce text, which previously provided no way for a computer to analyze its semantic or syntactic meanings, into vectors that can be compared against each other to provide meaning and practical value. For instance, Ibrahim et al. [5] used word embedding techniques in order to enrich consumer health vocabularies by mapping health jargon into layman terms. This was done by finding jargon that appeared in similar contexts to the layman terms and deducing that these words have the same general meanings. Another example of the use of word vectors is that of query expansion. In this case, as shown by Kuzi et al. [6], word embeddings can be used to find similar words to those included in a search query, so as to provide a wider range of documents for a user to select from once the search query runs.

However, in order to be usable, the vectors produced by these word embedding algorithms must be highly accurate so as to produce meaningful results. One test used to determine this accuracy is the synonym task, a similarity analysis. This paper will compare three algorithms – GloVe, WOVe, and Word2Vec – to see which produces the most accurate word vectors in a set context.

1.2 Objectives

The main objective of this research was to compare three different word-embedding algorithms in a similarity analysis in order to determine which was the most effective at generating correct synonyms for a variety of target words. This evaluation would help provide insight into which of the algorithms perform better in certain circumstances, allowing future researchers to determine which would be the best to use for their research goals. This comparison would be completed in two ways.

First, the overall number of correct synonyms generated by each of the three algorithms would need to be compared. In order to show that the word vectors generated by these three word embedding algorithms are accurate and reliable, one would need to show that they would be able to generate the same words that a human would when tasked with finding synonyms for a certain word.

Second, once the synonyms have been generated, the algorithms need to show that their generated synonyms are more accurate than the others'. This is done by comparing the positions of human-ranked synonyms to the ones outputted by the algorithms.

2. Related Works

The main idea behind using word embeddings to find synonyms is that of the distributional hypothesis, the idea of which was originally described by Harris [3]. In essence, the distributional hypothesis states that words that appear in similar contexts are more likely to be similar to each other. Applying this hypothesis is one of the main ways that word embedding algorithms have been tested for accuracy, since it is achievable for humans to find instances where this will occur and provide the evaluation datasets needed so that the word vectors generated by the algorithms can be compared against real data.

In the past, multiple researchers have looked at the various existing word embedding algorithms and compared them against each other to see which has had the best performance in a variety of circumstances. Wang et al. [13] compared six different word embedding algorithms, including GloVe and ngram2vec, in a variety of tests ranging from word analogies to concept categorization and outlier detection, with multiple different algorithms performing better at each. In another paper, Sutton and Cristianini [12] compared GloVe, Word2Vec, and fastText, on the task of capturing concepts from randomly generated word lists.

This research expands on those past works by comparing two of the generally highest performing algorithms, GloVe and Word2Vec, with a more recently released one, WOVe, a GloVe modification that has not been fully compared against them. Although WOVe has been described to outperform GloVe in the analogy task (using word vectors to find the missing word in an analogy), it has not been tested in the other popular natural language processing test – the synonym test (using word vectors to find correct synonyms for a word). Because of this, this paper aims to provide insight into this particular performance comparison.

3. Approach

3.1 Overview

There are a multitude of word embedding models available that train word vectors for language processing. One popular method, Word2Vec, uses the continuous-bag-of-words (CBOW) approach to generate word vectors based on context word probability [8]. Another method, GloVe, uses a log bilinear approach to reduce the dimensionality of word-word cooccurrence matrices [10], resulting in a word vector that can be trained quickly and has been shown to be highly accurate. Recently, Cox proposed a new method, WOVe, that expands on the functionality of GloVe by taking distance from a target word into account and combining multiple word vectors together for a more accurate word vector representation [2]. However, Cox mainly focused on comparing the performance of WOVe to GloVe when generating words for analogies – the analogy task. Because of this, we looked at a comparison of the performance of the three algorithms – Word2Vec, GloVe, and WOVe – in a similarity analysis to evaluate their effectiveness at the synonym task.

3.2 Method 1 – Word2Vec (using Continuous-Bag-Of-Words)

The first word embedding technique being looked at in this paper is Word2Vec, a prediction-based method where the CBOW approach is used to generate word vectors. CBOW is a neural network approach to the word embedding problem that works by providing probabilities of how likely other words will be in the context of the target word and then putting these weights into a word vector representation. For each target word, all the context words around it are processed through a shallow neural network that takes the words in, weights them depending on their arrangement and relation to the target word in a hidden layer, and then outputs the target word's word vector. The advantages of this CBOW approach is that neural networks use much less RAM than giant cooccurrence matrices while still generating accurate word vector results based on the probabilities generated. However, this kind of method can take a long time to train word vectors from scratch due to the complexity of the neural network.

3.3 Method 2 – Global Vectors for Word Representation (GloVe)

The next word embedding technique being compared in this paper is GloVe [10], a count-based method proposed by Pennington et al. in 2014. In essence, the algorithm goes through a series of steps, shown in Figure 1, that counts the number of times each word in a corpus of text appeared, generates a word-word cooccurrence matrix of those words, shuffles the positions of the items in the matrix to provide some randomness, and trains the resulting cooccurrence matrix on a weighted least squares model to provide the resulting word vectors. The most important steps in this process are the creation of the cooccurrence matrix and the weighted least squares model.

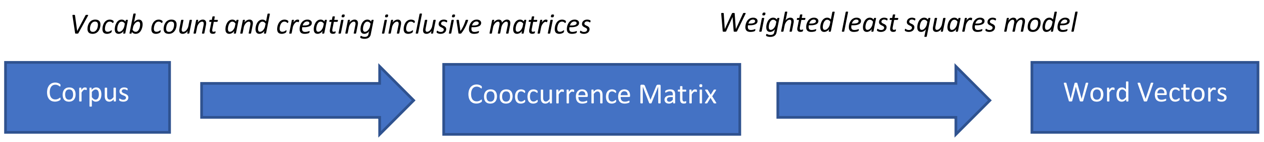
Vocab count and creating inclusive matrices
Weighted least squares model
Corpus
Cooccurrence Matrix
Word Vectors

Figure 1: Overview of the steps GloVe takes to produce word vectors

This model is trained using a local context window (e.g. +/- 5 words) to look at the words surrounding a target word and put them into a word-word cooccurrence matrix using inclusive matrices. For instance, consider the following sentences to be the corpus being analyzed:

“I have a dog that sits on my lap.”

“At the park, a dog enjoys running around.”

“Animal shelters have dogs to adopt.”

If we are trying to generate an inclusive matrix for the word “dog” with context size 2, then we would find the number of times each word within 2 words of the word “dog” appeared. This is represented in Table 1.

	have	a	that	sits	park	enjoys	running	shelters	to	adopt
dog	2	2	1	1	1	1	1	1	1	1

Table 1: An entry in an inclusive cooccurrence matrix with a context window of size 2 for the above sentences

Once this cooccurrence matrix has been created and includes all the words in the corpus, then the positions of the items in each entry get shuffled so as to not be in the same positions they were in the corpus. Since a general pitfall for this kind of approach is that cooccurrence matrices can get to be very large very quickly when taking into account all the possible ways words can be combined in sentences, GloVe goes one step further and performs a dimensionality reduction of the cooccurrence matrix by removing contexts that very seldom occurred, as well as traditional connector words (such as “the”, “and”, “a”, etc.) since these are not the words we are trying to derive meaning and word vectors from.

The last step is the weighted least squares model where the final word vectors are generated using the formula in Equation 1, where X_{ij} is the number of times that the i^{th} and j^{th} word occur together, w_i is the word vector of the i^{th} word, $\tilde{w}_j$ is the context word vector of the j^{th} word, and b_i and $\tilde{b}_j$ are biases for the w_i and $\tilde{w}_j$.

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - log X_{ij}\right)^2$$

Equation 1: The weighted least squares model used by GloVe to generate word vectors. Further explanation in the GloVe paper [10]

GloVe became a popular word-embedding model to use due to how easy and quick it was to train on an inputted corpus, as well as the fact that it outperformed most other models at the analogy task at the time, including Word2Vec.

3.4 Method 3 – Word Order Vectors (WOVe)

The final word embedding technique evaluated in this paper is WOVe [2], a modification upon GloVe proposed by Cox in 2019 that was able to improve GloVe's effectiveness in the analogy task by 9.7%. While GloVe does use word-weighting based on those words' distance from the target word when creating the word vector, it does so by generating inclusive matrices. For an inclusive matrix, all words from the target word to the edge of the context window are considered and weighted according to their distance, resulting in a singular vector. WOVe proposes going one step further and generating exclusive matrices, where the words at each step away from the target word to the edge of the context window are considered separately, resulting in multiple vectors being created and then combined together in post-processing. For example, if the same sentences from 3.3 are used, then for a local context window of size 2, the following two partial exclusive matrices are generated.

	a	that	enjoys	have	to
dog	2	1	1	1	1

Table 2: An entry in the partial exclusive cooccurrence matrix with a context window of size 2 and a range of 1

	have	sits	park	running	shelters	adopt
dog	1	1	1	1	1	1

Table 3: An entry in the partial exclusive cooccurrence matrix with a context window of size 2 and a range of 2

Each of these exclusive matrices would then go through the weighted least squares portion of GloVe before being combined together to produce a more accurate representation of each entry's word. While this does result in the size of the final word vectors being dramatically increased from m (the size of original vector) to m*n (the size of the original vector times the size of the context window) and a much increased time required to train the vectors, it did also result in an improvement over GloVe's already impressive results.

3.5 Testing Approach

In order to perform the similarity analysis, the overall goal would be to compare a list of ranked synonyms for a target word from the hard-truth synonym datasets described in Section 4.1 to a list of ranked synonyms for that same target word from the algorithms. Then, one would be able to see how many correct synonyms the algorithm generated, as well as how similarly ranked they were to the hard-truth dataset.

To output this list of ranked synonyms from the algorithms, the first necessity would be to generate word vectors from the pre-processed text corpus. Then, a target word from the datasets could be fed in with a program outputting the ten-closest word vectors to the word vector for that target word by using cosine similarity. Cosine similarity is a method of comparing how similar two vectors are by putting them through the following formula, which will output a value from -1 to 1 based on how similar the two vectors are, with values closer to 1 being designated as more similar:

$$\cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{\|a\|\|b\|}$$

Once this list has been generated, it can be compared to the list of synonyms of the same target word in the hard-truth dataset. To find the number of correct synonyms generated, one only needs to count how many of the synonyms in the hard-truth list are in the algorithm-generated list. Then, to find out how correct these found synonyms were, the positions of the synonyms in the two lists need to be compared. The closer the positions of the synonyms in the two lists were to each other, the better the algorithm was doing a better job at producing correct synonyms.

3.6 – Metrics

For the first test in this paper, we evaluated the total number of correct synonyms generated by each algorithm. We ran each algorithm in each context window +/- 1 through +/- 5, and compared the top ten synonyms generated to the hard-truth synonyms provided in our dataset. For every synonym that matched, regardless of its position, we added 1 to the total number of correct synonyms generated by that algorithm at that context window. Then, once all the algorithms had been run at that context window size, the summed results could be compared against each other (as seen in Table 4).

For the second test in this paper, we evaluated the average rank of the word embedding algorithms. To do this, we started with the same process as in the first test by generating the top ten synonyms for each target word and comparing them to the hard truth datasets to see how well they matched. Then, to find out how correct these found synonyms were, the positions of the synonyms in the two lists were saved and compared. For example, if the human-ranked list had three synonyms for a target word at positions {1, 2, 3} out of a list of ten, and the algorithmically generated list had those same three synonyms at positions {5, 7, 9} respectively, then the average rank of the algorithm for that target word would be the average of the differences between the two lists:

$$\frac{(5-1)+(7-2)+(9-3)}{3} = \frac{15}{3} = 5$$

If a synonym existing in the hard-truth dataset was not found in the list of the top ten synonyms generated by the algorithm, then it would be given a difference of 10 (1 more than the maximum possible between found synonyms) when calculating the average target word rank. Once the average rank of each target word had been calculated, then the overall average rank of the algorithm for that context window would be the average of all the average target word ranks. This was taken by summing all the average target word ranks and dividing by the number of target words.

4. Evaluation

The overall goal of the similarity analysis evaluations was to determine which of the three word-vector algorithms performed the best at generating correct synonyms (where correctness is determined by comparing algorithmically generated synonyms to human ranked ones). Word vectors can provide a good way to do this, and are why we chose to use three different word-vector algorithms, since one can compare the word vector of a target word to the word vectors of other words, with the most similar being more likely to be a synonym of the original word.

4.1 Datasets

In order to generate the results described in this paper, we used an 8.7 GB corpus of text from a 2019 Wikipedia text dump (the same corpus used for the WOVe paper in order to assure consistency). This text corpus was first tokenized to remove all XML tags, new lines, and punctuation. Then, all the remaining text was lower-cased and each word was separated by a single space. This was once again to ensure consistency with the text corpus expectations of the GloVe, WOVe, and Word2Vec algorithms.

For the testing, five different synonym datasets were used as the hard-truth datasets to compare the three algorithms to. These datasets were ones that are commonly utilized in the field of similarity analysis, containing a list where each element included a target word, a prospective synonym, and a human generated ranking on how similar those two words were. To use them in this research, we sorted each of them by target word, and then sorted that by descending human-generated ranking of the pair of words. So, if a target word had multiple pairings associated with it, they would be listed in order from most to least similar. The datasets used were as follows: Miller-Charles 28 dataset [9], Rubenstein-Goodenough dataset [11], WordSim353 dataset [1], Stanford's Contextual Word Similarities dataset [4], and the Stanford Rare Word Similarities dataset [7].

4.2 Method

In order to generate the results found in this paper, we first took our pre-processed Wikipedia dump as a corpus and gave it to each of the algorithms in order to have them generate word vectors. We chose to make these word vectors have a size of 50 in order to be consistent with both each other and the way that the data in the WOVe paper was gathered. Then, once the vectors had been generated, we carried out the similarity analysis tests on them, using 5 different context window sizes, from +/- 1 to +/- 5, for each of the algorithms. Overall, 2,480 target words (from the similarity datasets) were used during each run.

4.3 Results

The results generated from these methods can be found below in Tables 4, 5, and 6. For the number of found synonyms, Word2Vec initially performs better than both GloVe and WOVe, but after the context window increases to a size of greater than +/- 2, WOVe pulls ahead and becomes the best algorithm for generating the correct synonyms. However, in the case of the average rank of the found synonyms, GloVe initially has the best performance, with an average rank of 1.437 off from the hard-truth dataset on a context-window of +/- 1. After that, WOVe has the best performance consistently, with the lowest average rank in the context windows of +/- (2,3,5).

	+/- 1	**+/- 2**	**+/- 3**	**+/- 4**	**+/- 5**
GloVe	285	332	357	385	386
WOVe	299	395	415	425	416
Word2Vec	378	417	404	406	396

Table 4: Total number of correct synonyms found per context window

	+/- 1	**+/- 2**	**+/- 3**	**+/- 4**	**+/- 5**
GloVe	1.437	4.777	3.947	3.835	4.054
WOVe	4.039	3.882	3.748	4.100	3.730
Word2Vec	4.680	4.727	4.248	4.081	4.635

Table 5: Average rank similarities to the hard-truth datasets of correctly found synonyms

	# of Found Synonyms	**Average Rank of Synonyms**
GloVe (+/- 5)	386	4.054
WOVe (+/-4)	425	4.100
Word2Vec (+/- 2)	417	4.727

Table 6: Comparison of best performing context window sizes for each algorithm

4.4 Results Discussion

Overall, the WOVe algorithm performed the best across the multiple context windows at producing the greatest number of synonyms, doing so in context windows {1,3,4,5}. This is likely due to the fact that the vectors created by the WOVe algorithm ended up being larger than the other two algorithms, since it combined multiple exclusive matrices together to form its word vectors. This extra context provided in the word vector could have allowed it perform better at generating word vectors that were slightly more similar to the target word. However, it is interesting to note that despite the fact that WOVe's overall performance at producing synonyms was greater than the other two algorithms, the number of generated synonyms in context window 5 was less than the number of generated synonyms in context window 4. This, supported by the fact that Word2Vec's number dropped as well and GloVe's upward trend slowed down dramatically at this point, suggests that a context window of 5 might be the point at which words no longer become important in being included in a word vector's context.

In terms of producing the most accurate synonyms, WOVe also performed better in the majority of cases than the other two algorithms, having the most accurate synonyms compared to the hard-truth datasets in context windows {2,3,5}. Once again, this is likely due to the larger word vector size, attributed to the combination of exclusive matrices together.

5. Discussion

5.1 Conclusions

The main goal of this paper was to examine three word embedding techniques – GloVe, Word2Vec, and WOVe – and compare them on the task of similarity analysis, or of generating synonyms. This was done by using datasets of human-ranked synonyms as a hard-truth and observing how, on average across multiple context window sizes, the algorithmically generated synonyms of these techniques compared to those. Both the number of correct synonyms generated and the similarity to the correct ranks of these synonyms were examined in this analysis (as described in Section 4). Overall, the WOVe algorithm performed the best at both tests, making it the algorithm we would recommend for other word-embedding researchers to use.

5.2 Performance Summarization of GloVe and WOVe

As described in Section 3, the GloVe algorithm is one of the most popular word-embedding models currently available and was the inspiration for WOVe (which modified GloVe slightly to increase its accuracy). This prominence originated from Pennington's 2014 paper that described a new method to generate word vectors, using a global log-bilinear regression model to quickly train co-occurrence matrices into word vectors that were highly accurate. Across multiple test cases in both the analogy and similarity tests, it frequently outperformed other algorithms such as Word2Vec and other vector log-bilinear models. Because of this success, Cox recently made modifications to the way GloVe generated its coocurrence matrices, mainly in the usage of exclusive matrices instead of GloVe's inclusive matrices, in hopes of showing that performance could be increased. In the creation of WOVe, Cox showed that the performance of GloVe in the analogy test could be improved upon, with the Direct Concatenation modifications providing a 9.7% accuracy improvement.

5.3 Future Work

Although these results provide some insight into the differences in effectiveness between the Word2Vec, GloVe, and WOVe algorithms' performances in a similarity task, there are still ways that this work could be improved upon in the future. For instance, it might be useful to compare the results seen in this paper (where the word vector size used was 50) to results generated with larger word vector sizes (such as 100, 200, etc.). This comparison could end up uncovering new and different results of the effectiveness of these three algorithms when they are being used in different contexts. This would be useful for researchers using text corpuses of different sizes and make-ups, since there might be one particular algorithm excelling in one scenario while another is better in a different one.

Another way to continue on with this work would be to use different text corpuses than simply a Wiki dump. Although the Wiki dump used in this paper was a very good tool to train word vectors on, due to its diverse collection of vocabulary and topics, it would be interesting to see how these algorithms performed when using a corpus of strictly one main topic, such as one of medical questions and responses. In a case such as this, it might be possible to get an even better understanding of the different strengths of these algorithms (the neural-network generated probabilistic context of Word2Vec vs. the dimensionality reduced cooccurrence matrices used by GloVe and WOVe) in more specific contexts.

6. References

[1] Agirre, E., Alfonseca, E., Hall. K., Kravalova, J., Pasca, M., Soroa, A. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of NAACL-HLT 2009*.

[2] Cox, B.S. "Incorporating Word Order Explicitly in GloVe Word Embedding," University of Arkansas Computer Science Honors Thesis, Fall 2019 (supervisor: Dr. Susan Gauch)

[3] Harris. Z. 1968. Mathematical structure of language. *Interscience tracts in pure and applied mathematics*.

[4] Huang, E.H., Socher, R., Manning, C.D., Ng, A.Y. 2012. Improving word representations via global context and multiple word prototypes. In *ACL*.

[5] Ibrahim, M., Gauch, S., Salman, O., Alqahatani. M. 2020. Enriching consumer health vocabulary using enhanced gloVe word embedding. In *arXiv*. 2004.00150.

[6] Kuzi, S., Shtok, A., Kurland, O. 2016. Query expansion using word embeddings. In *Proceedings of the 25th ACM International on Conference and Knowledge Management (CIKM '16)*. Association for Computing Machinery, New York, NY, USA, 1929-1932. DOI: https://doi.org/10.1145/2983323.2983876.

[7] Luong, M., Socher, R., Manning, C.D. 2013. Better word representations with recursive neural networks for morphology. *CoNLL-2013*.

[8] Mikolov, T., Chen, K. Corrado, G., Dean, J. 2013. Efficient estimation of word representations in vector space. *ICLR Workshop*

[9] Miller, G.A., Charles, W.G., 1991. Contextual correlates of semantic similarity. *Lang. Cogn. Process*. 6, 1–28.

[10] Pennington, J., Socher, R., Manning, C., 2014. Glove: Global vectors for word representation, in: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pp. 1532–1543.

[11] Rubenstein, H., Goodenough, J.B., 1965. Contextual correlates of synonymy. Commun. *ACM 8*, 627–633. https://doi.org/10.1145/365628.365657

[12] Sutton, A., Cristianini, N. On the learnability of concepts: with applications to comparing word embedding algorithms. in: *IFIP Advances in Information and Communication Technology*. http://dx.doi.org/10.1007/978-3-030-49186-4

[13] Wang, B., Wang, A., Chen, F., Wang, Y., Kuo, C.C.J. 2020. Evaluating word embedding models: methods and experimental results. in: *APSIPA Transactions on Signal and Information Processing*. http://dx.doi.org/10.1017/ATSIP.2019.12